REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

AD-A208 443

1b RESTRICTIVE MARKINGS: NONE

3 DISTRIBUTION/AVAILABILITY OF REPORT: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

5 MONITORING ORGANIZATION REPORT NUMBER(S): AFIT/CI/CIA-88-216

6a NAME OF PERFORMING ORGANIZATION: AFIT STUDENT AT University of Colorado

6b OFFICE SYMBOL (If applicable):

7a NAME OF MONITORING ORGANIZATION: AFIT/CIA

7b ADDRESS (City, State, and ZIP Code): Wright-Patterson AFB OH 45433-6583

8a NAME OF FUNDING/SPONSORING ORGANIZATION:

8b OFFICE SYMBOL (If applicable):

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:

8c ADDRESS (City, State, and ZIP Code):

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification) (UNCLASSIFIED)
SOFTWARE MANAGEMENT INDICATORS:
MANAGING THE RISK OF PROJECT MANAGEMENT

12 PERSONAL AUTHOR(S)
CAPTAIN COURT COLLINS ALLEN

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
THESIS/DXXXXXXXXXXX	FROM _____ TO _____	1988	77

16 SUPPLEMENTARY NOTATION: APPROVED FOR PUBLIC RELEASE IAW AFR 190-1
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer, Civilian Institution Programs

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL
ERNEST A. HAYGOOD, 1st Lt, USAF

22b TELEPHONE (Include Area Code)
(513) 255-2259

22c OFFICE SYMBOL
AFIT/CI

DD Form 1473, JUN 86

SOFTWARE MANAGEMENT INDICATORS: MANAGING THE RISK OF PROJECT MANAGEMENT

by

CAPTAIN COURT COLLINS ALLEN
A.A., Colorado Mountain College, 1977
B.S., Fort Lewis College, 1979

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Management Science
and
Information Systems
1989

This thesis for the Master of Science degree

by

Captain Court Collins Allen

has been approved for the

Department of Management Science

and

Information Systems

by

Kenneth A. Kozar

Kenneth A. Kozar

Carroll W. Frenzel

Carroll W. Frenzel

Date Jan. 6, 1989

Allen, Court Collins (M.S., Management Science and Information Systems)

Software Management Indicators: Managing the Risk of Project Management

Thesis directed by Associate Professor Kenneth A. Kozar

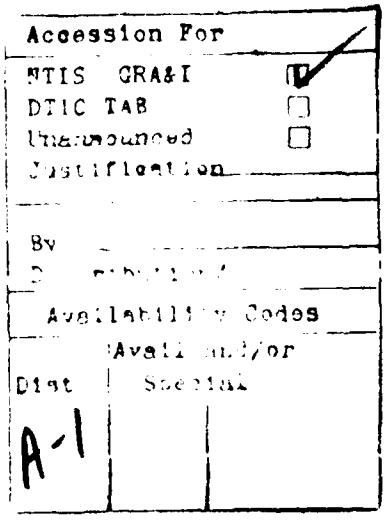
Managing software projects continues to be a high risk activity. Projects often end up out of control and result in cost and time overruns. Projects could be better managed if managers were forewarned that a project may be out of control before it is actually out of control. With advance notice of changing risk factors, project managers can take appropriate action to bring the project back on track and reduce project management risk's impact.

To create this early warning system, an effort to develop a suite of software management indicators has been evolving. Software management indicators are equivalent to an information system for software project management that predict project development process quality or "health." This will allow managers manage risk associated with software development efforts.

Software management indicators will be discussed and explained. Since much of this work has been done by government contractors in the United States, the focus will be to explain the use of methods developed in this environment . Exploration of experiences with these indicators will be shared. Qualitative research done by querying promoters and users of these techniques will explain what software management indicators are, why they should be used, when they are appropriate, and implementation hurdles that need to be cleared.

DEDICATION

To my incredibly understanding and loving family; my wife Peggy Ann, my daughter Courtney Jane, and my soon to be born son, whose conception interestingly coincided with the inception of this project.



ACKNOWLEDGEMENTS

My sincere appreciation goes to Dr. Kenneth Kozar, whose "humanized" and "discovery" approach to problem solving and learning made this project a profound experience. His patience and subtle, unassuming guidance over the last several years has allowed me to professionally grow and mature beyond expectation. Additional gratitude is extended to Dr. Carroll Frenzel and Dr. R.C. Mercure for their reassuring and invaluable guidance and their unselfish time in reading and criticizing this work.

DISCLAIMER

The views, opinions, and/or findings herein are those of the author and do not necessarily reflect those of the United States Department of Defense. This document should not be construed as an official Department of Defense position or policy unless so designated by other official documentation.

Furthermore, whenever "man", or "men", or their related pronouns appear either as words or parts of words (other than with obvious reference to named male individuals), they have been used for literary purposes and are meant in their generic sense.

CONTENTS

FIGURES

Figure

CHAPTER I

INTRODUCTION

Today's software development project managers are faced with the overwhelming task of effectively managing the evolution of today's information technology. They must integrate considerable past investments with rapidly changing future technologies. Because of the extraordinary demand for software, project managers are torn between the desire to develop quality systems and the conflicting need to bring new systems on-line quickly.

Because of these conflicting demands, managing software projects continues to be a high risk activity. Project managers are faced with increasingly complex system problems requiring complex system solutions fraught with risk. Their actions must consider the complex inter-relationships and wide variety of potential impacts on an overall system. Frequently, the inherent risk will cause them to experience loss of control, resulting in cost and time overruns. Successful projects result from effectively controlling this complexity along with numerous factors that introduce risk and work against project success.

Consider the following statements that profess a software project management control problem. Despite contrary examples of large projects that may be considered successes, the "testimonials" raise serious doubts about our ability to manage the scope and

complexity of many of today's software development efforts.

> Although every DP [information technology] project faces technical difficulties, they are not the major cause of project failures. The truly dramatic failures are due to inadequate or inept project management, which allows a project to run out of control like an unsteered car on an icy road. With no one in the driver's seat, the project crashes, its objectives unmet and its team left to scramble from the scene of the disaster in the humiliation of defeat. (Page-Jones, 1985, p. 61)

Ken Orr (1985) in offering options to manage what he calls the "Software Crisis" writes:

> Information systems managers today face four monumental problems: too much to do, too little time to do it in, too few good people and too many options. Industry experts say there's a software crisis, but in reality what exists is a software management crisis.
> (p. ID-2)

Comments by a recent Defense Science Board Task Force (1987) on military software add further credence to these assertions:

> Software plays a major role in today's weapon systems. The "smarts" of smart weapons are provided by software. It is crucial to intelligence, communications, command, and control. Software enables computerized systems for logistics, personnel, and finance. The chief military software problem is that we cannot get enough of it, soon enough, reliable enough, and cheap enough to meet the demands of weapon systems designers and users...We are convinced that today's major problems with military software development are not technical problems, but management problems. (p. 6)

Not to belabor the point, Cooper (1978) provides more insight into the deficiency of software project management:

> Perhaps this is so because computer scientists believe that management per se is not their business, and the management professionals assume that it is the computer scientists' responsibility. (in Abdel-Hamid and Madnick, p. 2)

The fact there are so many presumed successful computer

systems in use today may tempt one to minimize these assertions. It could be said that even though there is an aura of success about these systems, they probably had problems and could have benefited from better software development management.

These comments may suggest that software development in some "non-military" environments has reached a level of complexity found in many defense systems. If so, there consequently is a need for more managerial sophistication and control. In other words, old software project management techniques, when used, seemingly are no longer working and must be improved to parallel advances in software development "technology".

Poor software quality due to these alleged software management deficiencies can result in major losses of money, property, or life. The tiniest software "bug" can "crash" the biggest of systems. Complex accounting, medical, real-time and military systems are especially vulnerable to these quality defects (Davis, 1987).

Poor project management control has resulted in numerous cases of software project cost overruns, performance shortfalls, and schedule delays; all ingredients for project failure and poor software quality. While it is tough trying to control the quality of any complex project, doing it for these increasingly complex software projects seems to be even harder.

With these critical problems and consequences, a big question arises. How can we use today's management approaches to manage the complexity and project risk forecasted for the future which includes

the Strategic Defense Initiative (SDI), a project of greater magnitude than previous systems projects?

This study will look at a process control/quality assurance and risk management approach that utilizes software management indicators to provide early signals of an out of control software development process. Understanding and using this "total quality control" and risk management process could ultimately enhance software quality, timeliness, and fulfillment of expectations at optimal cost.

The Need for an Early Warning System

What is necessary is an "early warning system" providing the information, "alarm signals", critically needed to continually monitor and control the software development process and to assess the project's overall "health". These alarm signals are "symptoms"; observable phenomenon which result from "causes" and arise from and accompany a "defect" in the software development process. A "remedy" is a change that can successfully eliminate or neutralize a cause of the defects. (Juran, 1980, p. 104).

This "early warning system" should be a part of an overall Project Management Information System (PMIS), or in another view, a Quality Information System (QIS). Whatever it may be called, with software becoming so critical to over-all system success, project managers must carefully track the quality of the development process and continually re-assess its dynamic risk factors. The "process" in the software development sense, is merely the software development

methodology being employed.

Informally, one might think of early warning indicators or symptoms as "feelings" or intuitions that things are or are not going as planned, that things are out of control. In a more formal sense they are measurements of attributes that when synthesized with historical information and experience give one a more "quantifiable" information for a notion that things aren't right.

Many complex software projects are managed with sophisticated project management and software engineering methodologies like work breakdown structures, PERT networks and prototyping. These approaches provide management control feedback. Many times the feedback from these methods isn't early enough and frequent enough nor appropriate to assess whether a project is out of control before it is too late.

The impact of unplanned surprises is particularly threatening. A project in control is one in which the number of surprises along the way is minimized (DeMarco, 1982, p.5). By establishing and tracking indicators that feedback early information that a project may be proceeding out of control, is one way to reduce project threatening surprises. With enough notice, many times project managers can take timely corrective action.

Essentially, the indicators 'raise a flag' for project management. The 'flag' says things are not going quite as planned and that something may be wrong and need changing. The indicators are merely a stimulus for management action. The goal is to know about the problems at early

"inchpebbles" rather than at a major "milestone".

Formal indicators are commonly used in economics. There are leading, lagging and coincident economic indicators that have been chosen based on their historical relationships with economic trends. By measuring and collecting economic data, these relationships are used to provide early warnings on how the current economic "process" will affect future economic performance. Indicators used for software development management are based on similar principles.

As a simple case in software development, consider a programming staff that has started coding a software system before there is a final design or even worse a complete requirements analysis. Most managers historically know there is a real good chance the project will have serious problems.

Suppose, more subtly, experience has shown a direct correlation between the number of compiler diagnostic messages and the number of modules that will not pass integration testing. If measured and statistically analyzed early enough, management can work to determine causes and initiate remedies long before the integration stage of the project. The sooner project management can do this, the less costly and time-consuming the changes.

As a more explicit example, software development experience has shown requirements changes will tend to cause growth in the amount of system hardware memory needed. It is common to specify at least 50 percent spare memory in systems to allow for future enhancements. Figure 1 is an applicable indicator that tracks the

increases in memory needed versus the amount of spare memory that was allowed for the hardware.

Note around contract month 18 as the memory utilization crept toward and surpassed the 50 percent spare threshold, management was alerted and took action. Management decided requirements changes needed re-evaluating

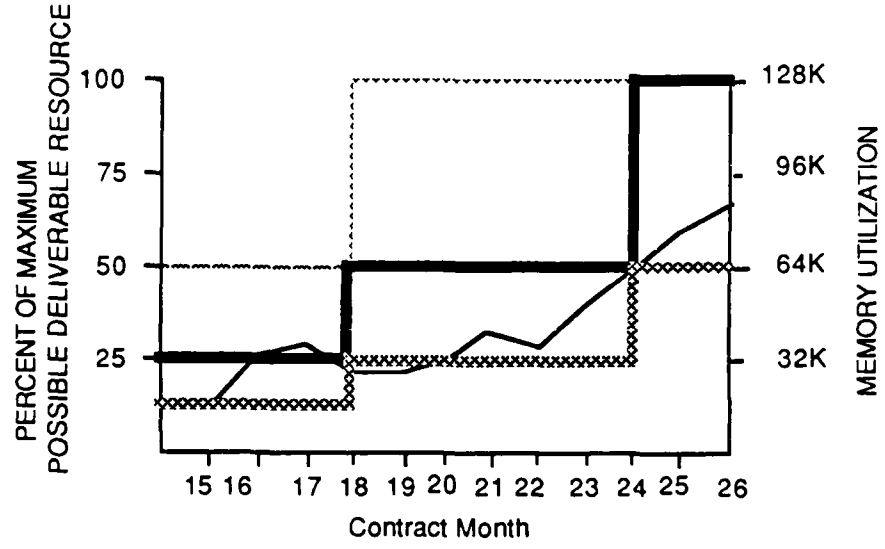


Figure (1): Software Management Indicator; Computer Resource Utilization, (Adapted from AFSCP 800-43, p. 5)

as indicated by the downward trend in the memory utilization plot. They also decided more memory be purchased as indicated by the quantum jump in planned deliverable resource at contract month 18. In both cases, management had better and early information from which to

operate and make critical project management decisions.

As shown, indicator information is normally depicted graphically. Figure 1 shows metric measurement data representing "actual" growth plotted against planned spare memory. From this summary, "instantaneous" data points can be represented "over time" and be extrapolated into the future to identify developing trends and routinely be fed into the risk management system.

The quality and timeliness of the managerial decisions depends in large part on the quality and timeliness of information. Decisions involve cost and cost estimating, schedules, product performance (quality, reliability, maintainability), resource commitment, project tasking, trade-offs, contract performance, and total system integration (Cleland and King, 1983). Projects are unique and so is the information required. It is therefore necessary to create a "quality control" and risk management information system that meets the project's unique needs. This is so project managers have the exact information they need to make informed, timely decisions that will keep a project progressing as originally expected.

These examples only briefly look at the types of things a software project manager can track informally or formally. What is needed is a concise, understandable, reliable, and manageable collection of these "key" management indicators. In a sense, these are "key success factors" that will be measured, tracked and assessed for use in these managerial decisions. Such a formal suite of software management indicators will be discussed in greater detail in the next

chapter.

The Process Control Approach

The ultimate yardstick by which the software is judged is whether it "works" and is "usable". Juran (1980) describes quality as "fitness for use" and a "defect" as any state of unfitness for use. Software containing bugs is defective. These defects cause a program to behave unpredictably and consequently render it unfit for use. The better the control of the "process" by which the software "product" is produced, the better the chance it will be fit for use. In other words, management should be as much concerned about the quality of the "process" as they are about the quality of the "product".

This is the process control approach to attaining quality. The principles of process control in engineering are known by a variety of names such as quality control, total quality control, zero defect, statistical quality control, quality assurance, and quality information systems.

When taken as quality assurance, Juran (1980, p. 516) feels "assurance" results from information that secures against some kind of disaster or risk. The information serves two purposes:

1. To assure the recipient that all is well, e.g., the product is fit for use; the process is behaving normally; and the procedures are being followed.

2. To provide the recipient with early warning that all is not well and that some disaster may be in the making. Through this early warning the recipient is placed in a position to take preventive action to avert a disaster.

Software Risk Management

Performance attributes of software determine how functional it is. Given enough time and enough money, most system performance requirements can be met. But, resource constraints demand that a balance between tradeoffs in schedule (time), performance (function), and budget (cost) be precisely managed by the project manager. Figure 2 shows how the required (ideal or desired) performance must compete and be traded-off with the conflicting goals of time and cost to meet an overall project target.

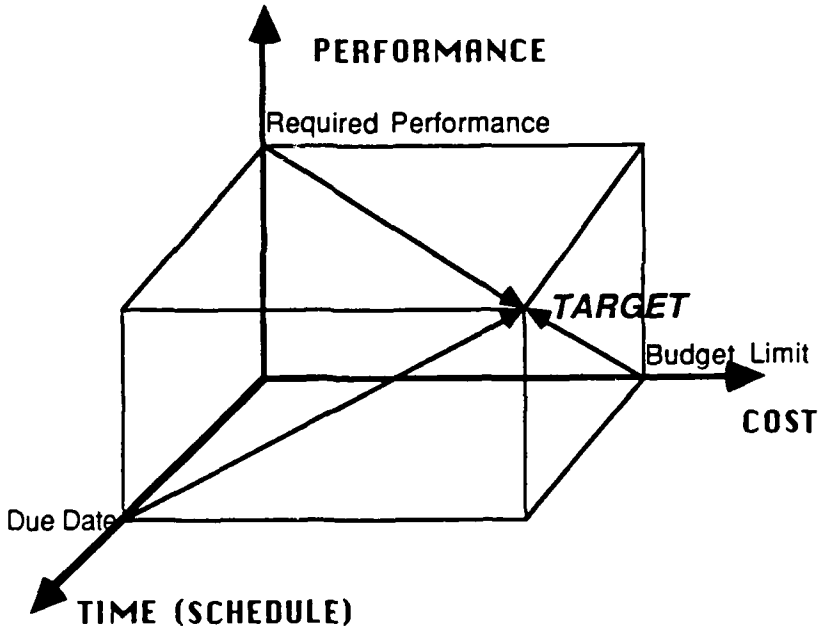


Figure (2): The Tradeoff of Performance, Cost, and Time Project Targets, (Adapted from Meredith and Mantel, 1985)

Most project managers are aware their projects involve a degree of ignorance and uncertainty to meet these cost, schedule, follow-on support, and technical performance objectives. Zmud (1980) feels to achieve more effective management of software development,

the amount of and ignorance uncertainty related to these objectives must be reduced within the project, and the information flow to decision makers confronted by the uncertainty must be facilitated. Their job is to identify and control the risk drivers resulting from the uncertainty.

Risk management is the way to reduce uncertainty and better manage the risk that threatens software system development. By systematically managing changing risk drivers, the threat to meeting a project's objectives will be optimized. Figure 3 depicts risk management as a "cocoon" which embodies the project. The process acts as a filter for software project management risk drivers while the project is "metamorphosing". Software risk management is an integral part of this overall systems integration approach.

As Figure 3 suggests, project risk can be partitioned into Performance, Cost, Schedule, and Support risk components. Each component of risk is influenced by risk drivers that cause the probabilities of cost, schedule, performance, or support risk to fluctuate significantly.

Figure 4 graphically depicts a formal software risk management system. This particular system was devised by the U.S. Air Force Systems Command (AFSCP 800-45). It provides the framework in which the use of software management

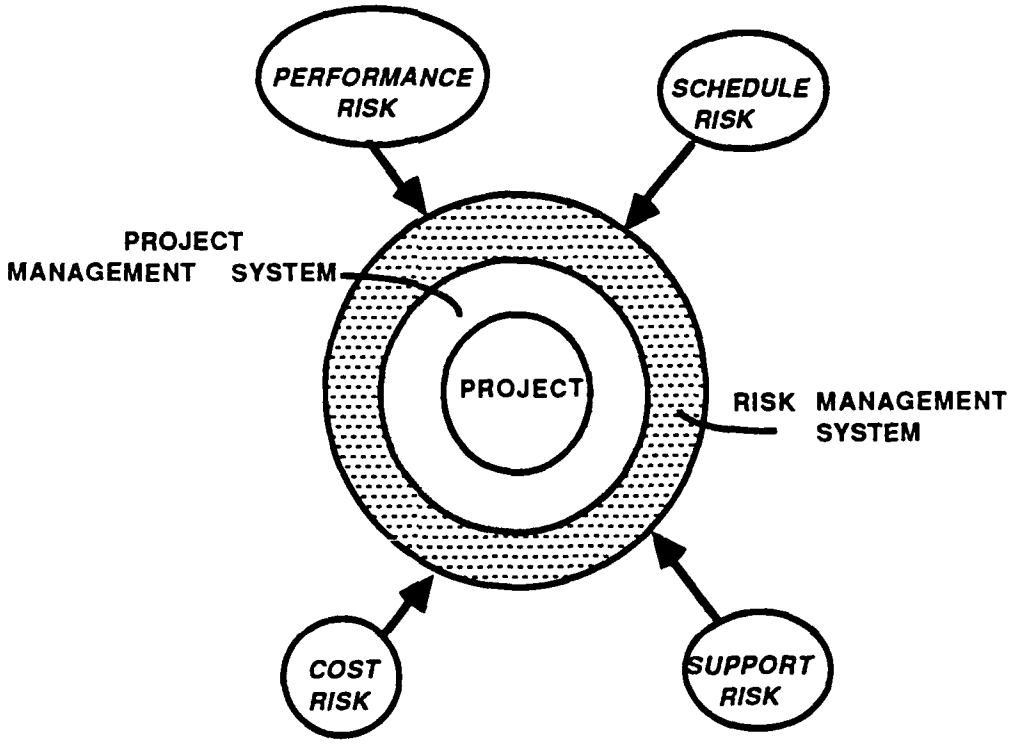


Figure (3): Risk Management as a "Cocoon" that Embodies a Software Development Project

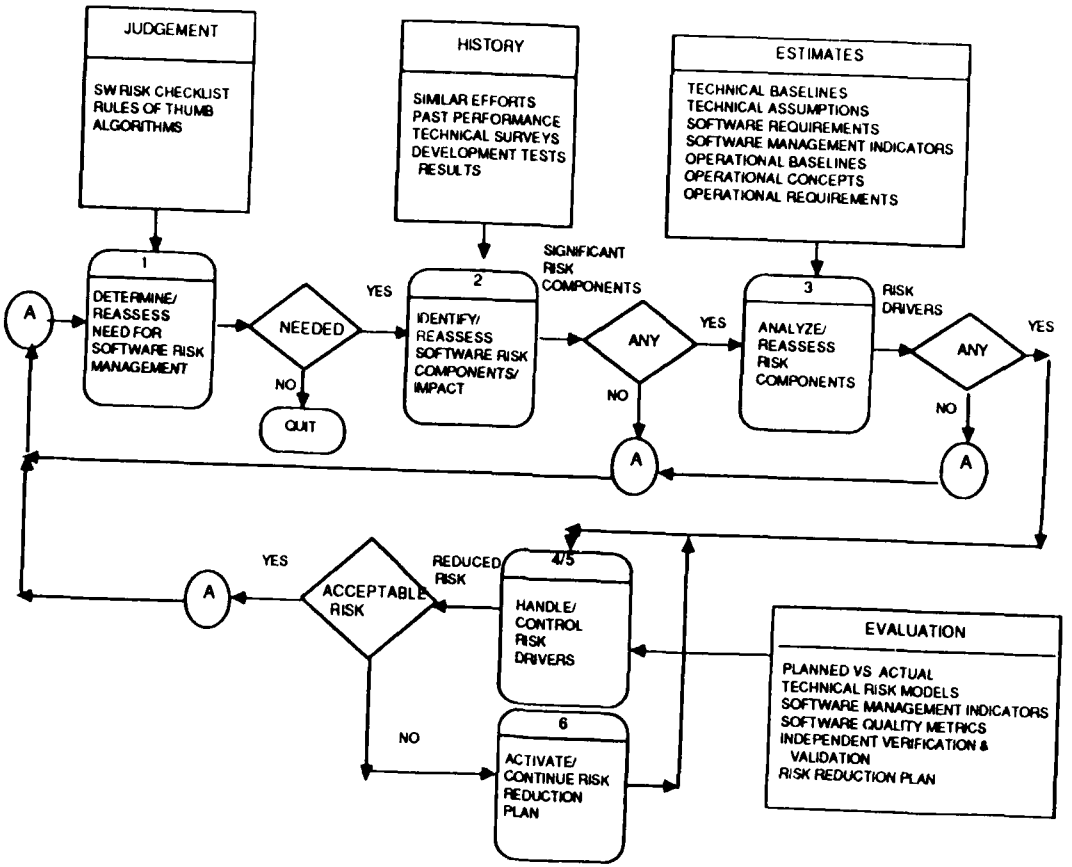


Figure (4): A Software Risk Management Process; A Framework from which to Employ the Use of Software Management Indicators and Manage Software Development Risk, (Adapted from AFSCP 800-45)

indicators are used. Those seeking more detailed information on the approach should see the reference. Briefly the processes are:

1. Determining whether there's a need for software risk management.
2. Identifying software risk components.
3. Analyzing risk component drivers.
4. Developing risk handling options.
5. Controlling risk drivers.
6. Iterative re-assessment throughout the system life cycle.

The software risk management system is designed to be an integral part of an overall-system risk management program. The system is an iterative process lasting throughout a systems development life cycle. It requires project managers to regularly assess baseline project objectives for risk that might increase the probability they won't be met.

The risk management system is used to gain perspective of the magnitude of the cause and effects of changes, to develop options, and to make decisions to enhance the visibility and control of a project's progress and final outcome. Visibility offers more insight and control reduces the difference between plans and reality. These objectives can be attained by improving communication and by using these more disciplined, systematic management techniques to proactively control expectations throughout the software life cycle. As shown in Figure 4, software management indicators are devised to provide critical information for the estimates and evaluations that drive the risk

management process.

Scope and Benefits of Research

This research examines the role of software management indicators in a quality control and risk management approach to enhancing management of large-scale software development projects. The complexity of this subject area is no less complex than the process of developing systems. Certainly, there are those whose expertise is far beyond that represented in this paper. An attempt is made to understand, integrate, and represent this expertise.

This effort will benefit the information management community by allowing wider dissemination of existing software management indicator knowledge. Recent work by the U.S Air Force that compiled a suite of Software Management Indicators (SMIs) will be used to represent the "state of the art" of this knowledge (AFSCP 800-43, 1986, ESD-TR-88-001). Although the work is biased toward military systems, there is applicability to all types of systems developments. It is hoped that this re-statement of the SMI principles will help clarify and stimulate their use and spur new directions and insights.

Preliminary research traced and established the historical and conceptual roots for using the suite of Software Management Indicators. It involved reviewing and abstracting literature on: the systems approach to management; systems engineering and project management systems; risk management systems; computer system development systems; quality/process control; and the conceptual

foundations of software metrics measurement.

The question is, does the use of SMIs help "grease the passage of software through the life cycle"? That is, does using SMIs as an "early warning system" in conjunction with software risk management help manage and control the complexity and risk in software development projects before they are out of control. If so, it is felt the ultimate quality of the software system will be improved.

An exploratory study was devised to qualitatively pursue this question. The study involved interviewing government and contractor authorities who have worked on large scale software development projects using SMIs. Interview questions were posed and solicited with a focus on validating the SMI use in quality control and risk management.

The questions also sought to identify policy and implementation issues, tangible and intangible benefits, problem areas, and recommendations for change. Although not the primary thrust of the research effort, pertinent findings that suggest further research needs and opportunities were also sought.

The next chapter will take a more detailed look at indicator principles and U.S. Air Force suite of Software Management Indicators. The suite will be used because it represents a concerted attempt to summarize many independent efforts and thoughts. Reviewing them should lead to better understanding of the current SMI concept. This understanding can then serve application of the concepts to other similar non-military environments.

CHAPTER II

SOFTWARE MANAGEMENT INDICATOR LITERATURE REVIEW

In this chapter, there will be a cursory review of indicator measurement fundamentals and precautions. Significant, topically related, prior efforts will be identified and briefly discussed. A detailed presentation of the U.S. Air Force SMI model, which synthesizes much of this work, will then follow .

Indicator Measurement

Metrics are measurable attributes or indicators that can support some quantitative aspect of a system for comparisons and evaluations. Many types of attributes can be measured. Once defined as measures for software management indicators, the metrics must have meaning in terms of "indicating" project process quality and changing risk trends. In this context it is key these indicators offer "assurance information" to management from which reasonable assessments and predictions are made of the performance of the software development process. To provide "early warnings" as a project progresses, metric measurement data is frequently collected and compiled.

DeMarco (1982, p. 54) distinguishes between result and predictor metrics. Result metrics may include total cost, total manpower, or total elapsed time. Predictor metrics would include "early noted"

indications that have a strong correlation to some later observed result, such as the number of early unplanned staff losses and future design schedule problems.

Analysis of metric/indicator data requires comparison with some "standard" that originates from project planning and risk analysis or from historical studies/experience. Today's software project planning will typically incorporate an estimating model similar in principle to COCOMO (Boehm, 1981). The model produces various estimates of how the software development project will progress over time. These "planned" estimates are used to compare and assess "actual" project performance. Similarly, historical studies and past experience provide some "guideline" or "rules of thumb" of what can be expected if the current project has qualities similar to those found in previous efforts.

Birrell (1985) gives examples of three cases that distinguish between indicators which are compared against plan and those which are compared against historical data.

1. An indicator that can be compared directly against the project plan.

> A good example of this is the measure of the progress of a work package (deliverable or module) against the project plan. This can be the difference between the actual date of completion of the work package and that required by the plan. By aggregating measurements of all of the project's work deliverables, there is an indication of the overall health of the project. A cost/schedule "earned value" system is typical of this type of indicator (See for example, Meredith and Mantel, 1985, pp. 296-297, 325).

2. An indicator that can be compared against plan by the use of historical data.

Consider the amount of design documentation as an indicator of possible computer memory problems. If there is an estimate contained in the project plan, then Case 1 would apply. Otherwise, the measurements might have to be converted to another measurement that can be compared to historical data. e.g. lines of code produced as a function of the amount of design documentation.

3. An indicator that can only be compared against historical data.

In this case consider a work package which is being tested. During testing each error detected is recorded. Once testing is done, the total number of errors detected per line of code is recorded as an indicator for the package. Assuming the project plan doesn't have a planned number of errors per work package, it may be compared to historical data available from past efforts.

Care in the Use of Metrics

Devising metrics to be measured is a difficult process. DeMarco (1982, p. 52) feels there are no such things as metrics that are independent of the conscious influence of project personnel. With this he restates Heisenberg's Uncertainty Principle as:

> Measuring any project parameter and attaching evident significance to it will affect the usefulness of that parameter.

In other words, if project members know what is being counted to measure a certain goal, they will tend to attach significance to it at the expense of others. DeMarco (1982, p. 58) extends this to say if you assign a dozen (often conflicting) goals, and only measure the performance of one, then, expressed as The Metrics Premise:

> Rational, competent men and women can work effectively to maximize any single indication of success.

A frequently cited programming experiment performed by Weinberg-Schulman (1974), offers a good example of this goal conflict.

When five different teams were given their own goal to maximize, it was found that they did it at the expense of the others. Boehm (1981, p.21) thinks the following conclusions can be reached from the experiment:

1. Programmers have very high achievement motivation.

2. Different software objectives conflict with each other in practice.

3. Successful life-cycle software engineering requires continuing resolution of a variety of important but conflicting goals.

Capers Jones (1986, p.5) feels the measurement of software programming has been the weakest link in the whole science of software engineering. When common metrics used for programming are analyzed under controlled conditions, he suggests three paradoxes result:

1. Lines-of-code measures penalize the conciseness of high-level languages and tend to be inversely proportional to productivity increases.

2. Cost-per-defect measures penalize high-quality programs and always are inversely proportional to quality improvements.

3. Ratios established for programming subactivities such as design, coding, integration, or testing often move in unexpected directions in response to unanticipated factors. That is they can be unpredictable.

Denicoff and Grafton (1981) argue metrics for the measurement of established science and engineering disciplines are based in the "laws" of physical nature (e.g. temperature, weight, area). In contrast,

measurements in computer science and software engineering depend on the ingenuity of the humans who can devise useful measures.

Denicoff and Grafton (1981) further suggest software engineering doesn't suffer from the number of metric measurement proposals. The problem with many metrics is they are commonly used out of necessity without benefit of a deep understanding. They feel that software and computer science as disciplines have more in common with economics, psychology, and political science than with the physical sciences. These disciplines also struggle with the problems of measurement which may stem from dealing with measuring human activities.

The availability of metrics, in itself, will not necessarily assure computer software development the status of a science. Denicoff and Grafton (1981) feel "quantitative" evaluation techniques are necessary for the evolution of software from an art to an engineering discipline.

The message should be clear. Measurement of the software development process is still immature and slowly evolving. The measure of most successful systems is whether "it works". Many of the commonly used metrics were developed to measure these more technical performance aspects of software; essentially its quality. There has been a strong tendency not to measure and assess the quality of managing the process life cycle toward attaining that performance.

Software Product versus Process Quality

Typical use of metrics in software quality control has been in the realm of attempting to measure and assess software "product"

quality. These types of measures represent the majority of prior effort and use of software metrics (see for example Birrell, 1985; Boehm, 1981; DeMarco, 1982; Gilb, 1977; Jones, 1986; and Perlis, 1981).

In Gilb's (1977) book, Software Metrics, he contends his work represents the first attempt to describe the then emerging technology of software metrics. He does acknowledge another preceding TRW Systems publication (TRW-SS-73-09, 1973), Characteristics of Software Quality, whose principal author was Barry Boehm, as a "remotely comparable" work. Boehm (1976) traced software metric work back further to a paper by Rubey and Hartwick in 1968. Other work by Boehm (1978, 1981) and Kosarajo and Ledgard (1974) prove to be the fundamental foundations from which many of today's indicator concepts have evolved.

Simultaneous work by DeMarco (1982) also uses the concept of metrics measurement in software development. It is here, factors of quality and indicators of quality are distinguished. Factors of quality include such measures as software correctness, flexibility, interoperability, portability, and reliability. Indicators of quality measure parameters such as design structure, defect density, test coverage, and documentation. Both categories attempt to measure the quality of the product.

The problem with these "traditional" measures of software quality is that they look at the product without considering the quality of its development methodology. Software Management Indicators (SMIs) what are used to measure this development process.

Unlike "hardware", successful management methods used to specifically assess the quality of an ongoing software development process have been informal and not widely known and disseminated. There has been an effort by the United States Department of Defense (DOD) to compile, synthesize, and standardize these methods with other metrics measurement principles. The DOD feels it must create a more formal early warning information system for managing their large and complex software development projects. The goal is to "formalize" ways that better predict project management "health" and to better manage the risk associated with software development efforts. Even though many of the techniques have informally been used for some time, until only recently (1982) has there been work to summarize and formalize them.

This U.S Air Force (USAF) Systems Command (AFSC) project surveyed government agencies and contractors to gather management techniques and "lessons learned" from successful software development efforts. A "suite" of Software Management Indicators (SMIs), that are an early warning information system, have evolved from this work (AFSCP 800-43). They represent an attempt to more formally help project managers evaluate the effectiveness of the software development management process.

Other related work of significance, Software Management Metrics, is an active implementation by the AFSC Electronic Systems Division (ESD) of some of the preliminary indicator work from the Air Force effort and contracted work by the MITRE Corporation (ESD-TR-88-

001, 1988). This most recent ESD/MITRE work is based on a previous MITRE publication, Software Reporting Metrics, authored by T.F. Saunders.

In the following section, there will be a detailed look at these SMI suites. Details and descriptions of the indicators have been liberally extracted from these sources (AFSCP 800-43, 1986; ESD-TR-88-001, 1988). From researching the software management indicator approach, it is felt this "model" represents the "state of the art". It best summarizes current, less formal and standardized, efforts across the defense industry.

The U.S Air Force SMI Model

The SMI efforts presented here were developed to improve the software management capabilities of DOD project managers and their counterparts in industry. The overriding goal is to improve visibility into and control of the software development process and thus better manage software development complexity and risk.

The following are other Air Force goals and objectives for the SMIs:

1. Have indicators that cover all phases of the software development life cycle

2. Present indicators that provide management visibility into critical success concerns.

3. Develop indicators easily used and analyzed to identify potential software development problems.

4. Create an effective risk management and control tool for top level project management. i.e. management at the overall system level.

The Air Force Systems Command has identified six overall SMIs, each with several supporting metrics. By no means does this model represent the definitive SMI model. It only represents one attempt to standardize many ways of approaching software management indicator measurement. Each indicator will be discussed in detail and supported with a graphic example and some "rules of thumb" regarding their interpretation and use. Before this detailed description, there will be a brief overview of the indicators and metrics as outlined in AFSC Pamphlet 800-43. They are:

1. Software Development Progress indicator metrics that track development progress by plotting the actual number of software units designed, coded and tested, and integrated against a project plan. (Note: A CSCI is a Computer Software Configuration Item; a "logical" grouping of software modules or Computer Software Units (CSUs), a "sub-system" that is managed as a whole). Metrics include: CSCI Unit Design Progress, Code and Test Progress, Integration and Test Progress, and Formal Qualification Test Progress.

2. Computer Resource Utilization metrics which intend to show whether the utilization of planned computer hardware resources is approaching pre-determined "spare"/growth limits. Metrics include: CPU Utilization, Memory Utilization, and I/O Utilization.

3. Software Manpower metrics attempt to show a project is proceeding according to planned staffing profiles. Metrics track numbers of Total Staff, Experienced Staff, and Unplanned Staff Losses.

4. Requirements and Design Stability metrics track the stability of requirements by measuring the numbers of requirements change proposals and software problem reports. The metrics are: Requirements Stability, and Design Stability.

5. Cost/Schedule Deviations provide a mechanism to track cost, schedule, and technical performance baselines. A commonly used methodology is C/SCSC (Cost and Schedule Control Systems Criteria for Contract Performance) "earned value" analysis techniques used in many government projects (see for example Meredith and Mantel, 1985, pp. 296-297, 325 or DOE/CR-0015, 1980).

6. Software Development Tools availability is crucial to understanding whether the proper software tools are available as needed for development and testing. Metrics track tools required and their availability dates in the Software Engineering Environment, and Software Test Environment.

SMI Descriptions

1. Software Development Progress indicators are designed to monitor the ability to maintain development and test progress. They measure the degree to which software unit design, code, test and integration is on schedule. Figure 5 depicts one way Software Development Progress metrics can be plotted.

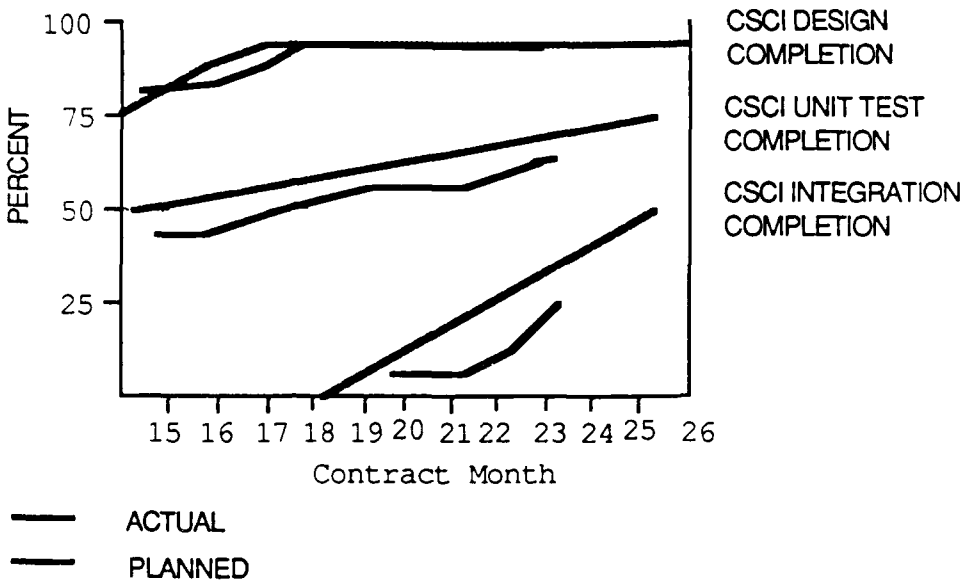


Figure (5): Software Development Progress Indicator, (Adapted from AFSCP 800-43)

1a. CSCI Unit Design Progress tracks the engineering design effort by tracking the percentage of computer software units for the CSCI that have been successfully designed. The metric measures:
(UNITS 100% DESIGNED)/(TOTAL UNITS PER CSCI)X100%

A software unit is considered successfully designed when it has been submitted for coding and unit testing. The graph plot of the indicator should show the planned and actual percentage of completed units versus a corresponding schedule. As is shown in Figure 5, a healthy program will experience an exponential growth in modules designed. This is reflected in the indicator's plot up until contract month 18 when the design process "levels off.". Erratic progress indicates problems with managing the design process and can be attributed to such things as worker experience levels, misunderstanding of requirements, and volatility of requirements.

1b. CSCI Unit Code and Test Progress tracks the unit coding and testing effort by indicating the percentage of CSUs for the CSCI that have been successfully tested. A software unit is considered successfully tested when it has met its design requirements. The metric for the indicator plots:

(UNITS 100% CODED:TESTED)/(TOTAL UNITS PER CSCI)X100%

The plot of the indicator should show the planned and actual percentage of tested CSUs against schedule. Figure 5 shows, for a normal program, the total number of sofware units that have passed unit test and the total number of units that have been integrated should be increasing linearly and constantly. Changes of more than 10 percent should be addressed. The example shows starting at contract month 19, there was a "plateau" in unit test completion for two months. This is a notable "early warning". The effect in the integration completion metric is shown as a plateau starting soon after the unit test process faltered.

1c. CSCI Integration and Test Progress tracks the CSU integration process to predict when the overall CSCI will be available for formal qualification testing. The indicator metric measures:

(UNITS 100% INTEGRATED INTO A CSCI)/(TOTAL UNITS)X100%

As with the previous indicators, it plots planned and actual percentage of completed integrated CSUs versus the schedule. Another metric plot, such as Figure 6, tracks the number of open software problems, Software Problem Reports (SPRs), and the "density" of the discovered errors (number per thousand lines of source code). Ideally, the number of new problems during the testing phase should not

be generating faster than old ones can be resolved. If the plot of open SPRs is positive, as is the case for the first few months in Figure 6, then problems are being identified faster than can be resolved.

Note, schedule slips in the completion of software testing may be estimated by observing the trend of the open SPRs. If the number of new and open SPRs is increasing as the test completion milestone is approached, then management will need to take action. An aggressive test schedule may be forcing testing and thereby be preventing the number of open problems from decreasing until all tests have been performed. This will probably lead to problems that get increasingly worse until all tests can be completed.

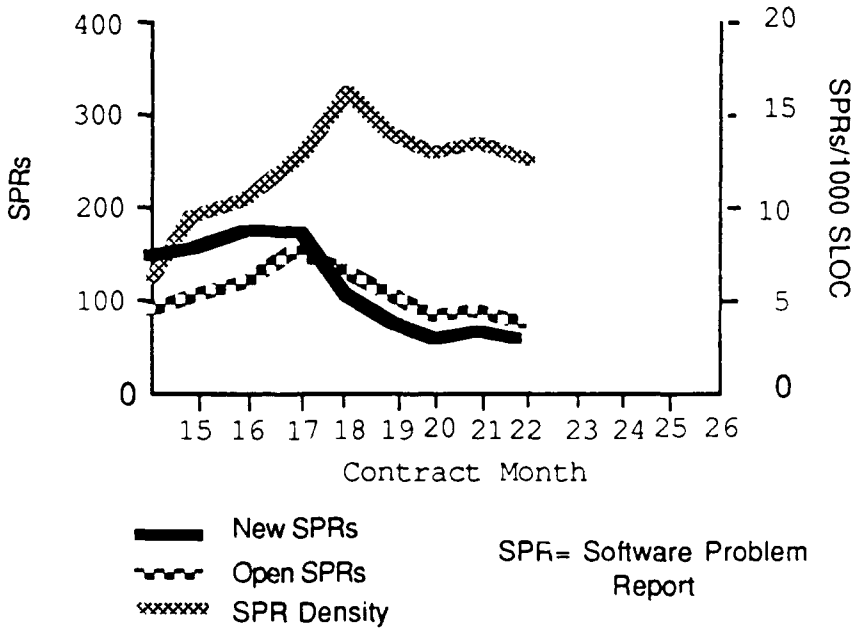


Figure (6): Software Development Progress: Integration and Test Software Problem Reports, (Adapted from AFSCP 800-43)

Software Problem Report density is considered "normal" when between 8 and 30 SPRs/1000 SLOC (Source Lines of Code). Too few

SPRs may indicate poor testing and too many indicate poor software code quality.

1d. Formal Qualification Test Progress plots the success and failure of the qualification tests conducted against the CSCI. It tracks the number of test procedures scheduled and the number of test s that passed. It should be "expected" that no problems will be uncovered during the testing, but commonly programs experience schedule slips and failed tests. It will give indications of whether the test program is going to take longer than planned.

Rules of Thumb. Some rules of thumb for this Development Progress indicator could include:

(1) Units tested and integrated should progress at a uniform, not sporadic, rate and according to plan. Sporadic development may be caused by factors such as overutilized development computers or under-experienced staff. This results in a high-pressure environment in which all software becomes due at once. For a "well-run" program, the indicator plots should each rise with a fairly constant slope.

(2) Completion of designed units should begin once about 25 percent of the total allotted time has passed. Experience has shown that increases in design effort may lead to reduced integration and test effort and higher quality products (ESD-WP-27367, 1987, p.28). In other words, if an attempt is made to save time on design and code activities, the savings will be more than offset by delays in testing and integration.

(3) At 80 percent of design, 50 percent of the design should be complete.

(4) Expect a linear completion rate during a code and the unit test phase. For example, at 25 percent of testing schedule, there should 25 percent completion of coded units.

(5) The number of SPRs is an indication of testing adequacy and of code quality. A range of 5-30 SPRs/1000 Source Lines of Code is "normal". In either case, the code may still contain a large number of undetected errors.

2. Computer Resource Utilization indicators provide tracking and assessment mechanisms that show the resource use, and spare capacity available for each computer hardware resource; CPU, Input/Output, and Memory. This ensures the software design will fit within the planned resources, and that adequate spare capacity is available to permit enhancements over the system's operational life.

The basis of this indicator is the computer resource utilization requirements outlined in the system specification. In essence the utilization baseline represents a "budget" of hardware resource available to the designers. Figure 7 shows a planned 50 percent spare for all three resources (Figure 1, in the previous chapter is another version of plotting this metric).

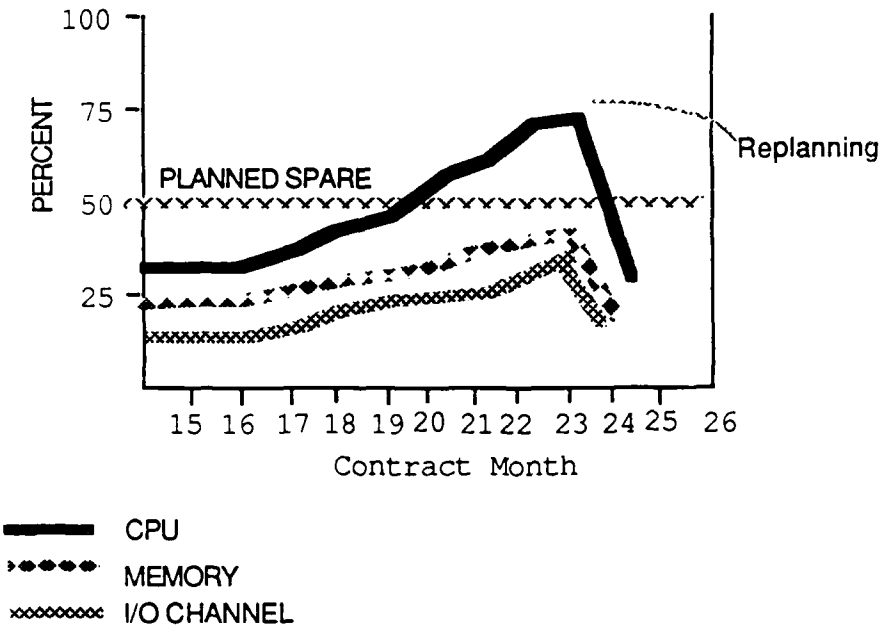


Figure (7): Computer Resource Utilization Indicator: CPU, Memory, and I/O Channel Resources, (Adapted from ESD, 1988)

Notice how each plot has a positive slope. Most development programs experience this upward creep in the estimate of resources needed. This creep is attributed to maturity and/or changes of requirements and design. In one study the average growth in utilization was 66 percent (ESD-WP-27367, 1987, p.9). If the creep exceeds some predefined spare, management action is required and might include the following: resource expansion, requirement growth curtailment, requirement reduction, software redesign, and possibly no action. The example shows at month 23 management decided to upgrade the hardware, perhaps by using a faster CPU.

Rules of Thumb.

(1) CPU, Input/Output, and planned Memory utilization should allow a minimum of 50 percent spare.

(2) Performance deteriorates, for example, when utilization exceeds 70 percent (for real-time applications).

(3) Resource utilization tends to increase with time. Plan for this expansion early in the software development cycle.

(4) Schedule, cost management, and adherence to development standards deteriorate dramatically as the spare resource drops below 10 percent. Software development costs and schedules increase dramatically as computer resource utilization limits are approached and optimizing forces design and coding changes.

3. Software Manpower is monitored to assess whether a project can keep qualified development staff committed as planned. Counts of unplanned personnel losses are maintained so that work force stability can be tracked. A program that begins with too few software personnel or that attempts to bring too many on board at the last moment (the "Mongolian herd" syndrome) is in trouble.

As Figure 8 shows, Total Staff, and Experienced Staff planned versus actual numbers are plotted. It is commonly felt that a successful project will use at least a ratio of total to experienced personnel should be near 3:1 and never exceed 6:1. An experienced individual is considered to have at least five years experience in software development and at least three years experience in development of applications similar to the current project. The example shows this criteria being met, with the highest ratio about 5:1.

The normal shape of the total software staff profile will show growth through the design phases, peak during coding, and testing

phases, and then gradually taper off as integration tests are successfully completed. The shape of the experienced personnel profile should be moderate during the initial stage of development, dip slightly during CSU development, and then grow somewhat during testing.

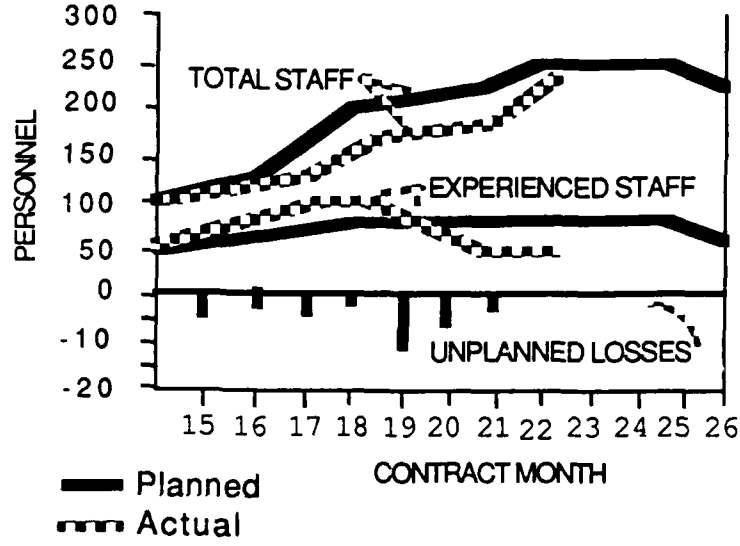


Figure (8): Software Personnel Indicator, (Adapted from ESD, 1988)

Rules of Thumb.

(1). The time required for software development depends on the staff-months delivered.

(2). Understaffing is an early indication of potential schedule slippage.

(3). Adding manpower to a late project will seldom help schedule problems.

(4). A program that is maintaining the staffing profile, but is experiencing a high personnel turnover rate, is not maintaining needed continuity among the design and implementation staff.

(5). Initial staffing levels should be at least 25 percent of the average staffing level.

4. Requirements and Design Stability indicates the "maturity" of the software requirements and design. It measures the number of requirements impacted by engineering change proposals (ECPs) and software action items (SAIs). Also, the number of SAIs can be tracked for each CSCI to assess the volatility of its design. The intent is to show the degree to which the software requirements have been defined using quantitative engineering measures to allow testing of those requirements. A lack of firm and well-defined requirements leads to uncertainty and changes that result in cost growth and schedule delays.

4a. Requirements Stability depicted in Figure 9, tracks the total number of software requirements as well as the cumulative number of changes to those requirements. Changes in the number of requirements, additions and deletions, directly impact the software development effort. Changes are expected in the early stages as details of the system design are defined and understood. At some point the software requirements must be "frozen".

There is an indication that this is occurring starting at contract month 23 in Figure 9. It will be assumed that this is appropriate for this development effort, but the longer this takes, the greater the impact on cost and schedule. More specifically, changes after the "critical design review" will probably have significant schedule impact, even if it is a requirement deletion.

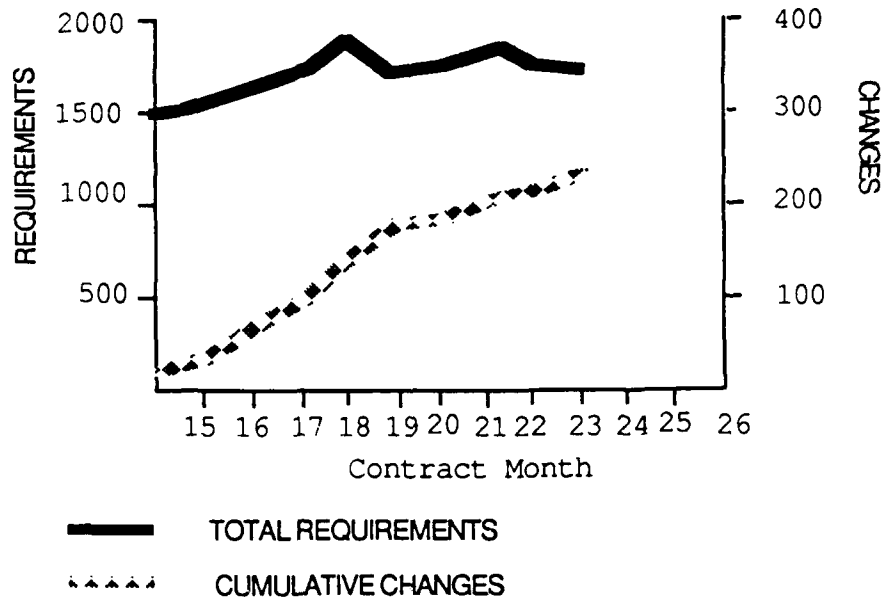


Figure (9): Requirements Stability Indicator; Software Requirements Changes, (Adapted from ESD, 1988)

4b. Design Stability can be measured by tracking the number of Software Action Items (SAI) resulting from design reviews. A SAI is defined as any discrepancy, clarification, or requirements issue that must be resolved. Theoretically, they represent inconsistencies between the requirements and the design or within the design itself that are discovered in the formal design review process. The plot of SAIs is expected to rise at each review (contract month 19) and then taper off exponentially. Note how Figure 10 depicts this.

Development efforts with clear and complete specifications will tend to experience less of a rise at each review, as will ones where the review process is inadequate. A well managed project will show a rapid "rate of decay" of these action items indicating ability to resolve problem issues.

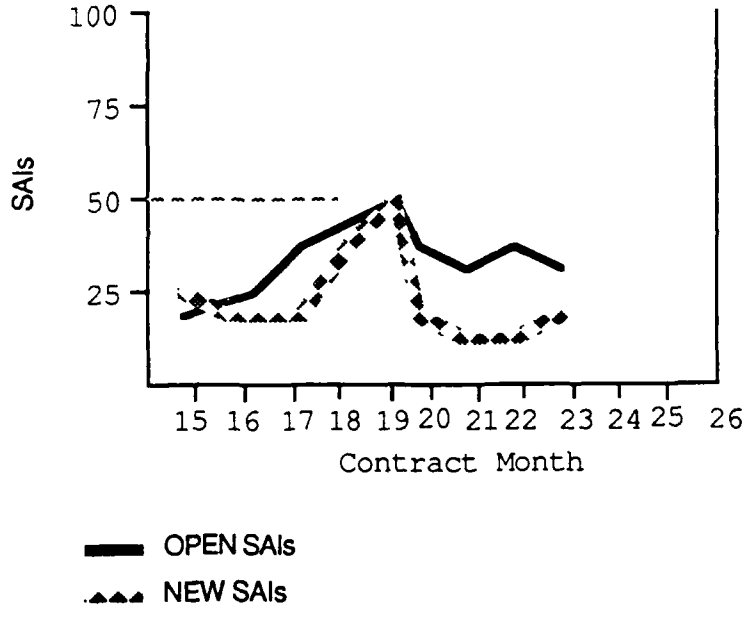


Figure (10): Software Requirements Stability: Software Action Items, (Adapted from ESD, 1988)

5. Cost/Schedule Progress tracks variances in the ability to maintain cost budgets and schedule plans. It uses data from a project's formal accounting system. A common system in government work is C/SCSC mentioned earlier in the chapter. This accounting/audit system is rather complex. Given this, there will be only a cursory attempt to describe the system. Those needing further clarification or detail should consult the previously mentioned references along with other government cost reporting and financial policy documents.

The C/SCSC concept measures work accomplished or "earned value" to have a better feel for true project progress. This is used because being at a certain point in schedule doesn't mean the planned amount of work has been actually performed. Similarly, if half of the budget has been spent, it doesn't mean half of the project is complete.

Earned value integrates cost and schedule plans with actual delivered work/function. This is so actual project costs are compared with meaningful values (the planned or budgeted value) for the work that has been accomplished, rather than with the work scheduled to be accomplished.

In doing this, a deviation results from accomplishing more or less (or different) tasks than were originally scheduled in a given time period. This separates this type of deviation from one that comes from the measurement of efficiency of performance for the work that is actually done.

Figure 11 graphically depicts the "earned value" approach. The budgeted cost of work performed (BCWP) is defined as the amount originally budgeted to do the amount of work that is now complete. The budgeted cost of work scheduled (BCWS) is the budgeted cost of work that was scheduled to be complete by now. The actual cost of work performed (ACWP) is the amount that was actually spent to accomplish the currently completed work.

From these values, the cost variance (BCWP-ACWP) and schedule variance (BCWP-BCWS) are used to calculate the Cost Performance Index (CPI) and the Schedule Performance Index (SPI). The CPI (BCWP/ACWP) is a measure of efficiency or productivity (i.e. cost performance). The SPI (BCWP/BCWS) is a measure of production or schedule performance. If the ratio for either index is greater than one then productivity and/or production is good.

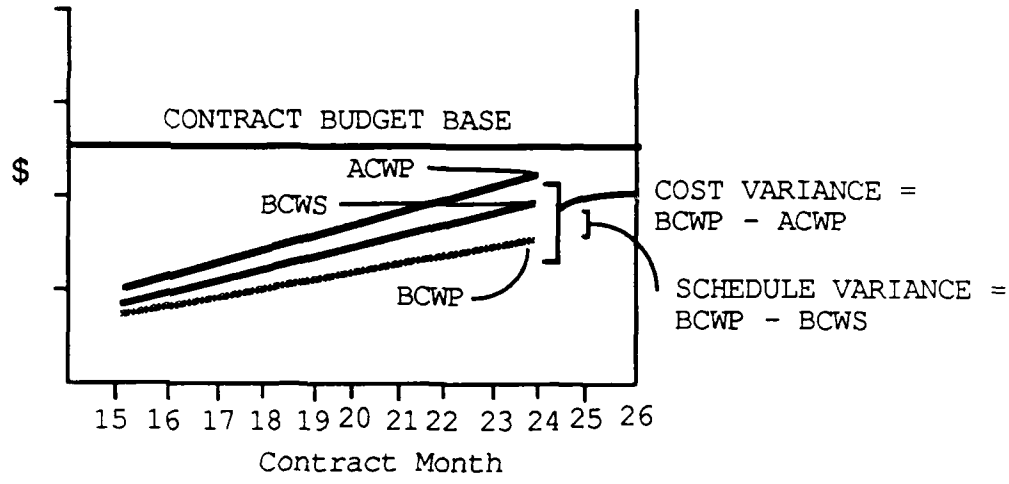


Figure (11): Cost/Schedule Indicator, (Adapted from AFSCP 800-43)

6. Software Development Tools indicator assesses whether the tools necessary to complete the software development process are available to the developer. Such tools may include testing programs, external environment simulation programs, compilers, and management information systems. It shows the degree to which the developer has established the software engineering and test environments to support the development process.

The metric used in this indicator, as shown in Figure 12, is the number of months available from when the software tools are required to when the tools are expected to be delivered. It is reasonable to expect some tools will be necessary. The indicator should show "positive margin" (tools 1 and 3) between when each tool is needed and when it will be available. If there is negative margin (tools 2 and 4), alternative

development approaches may need be identified in the risk management process.

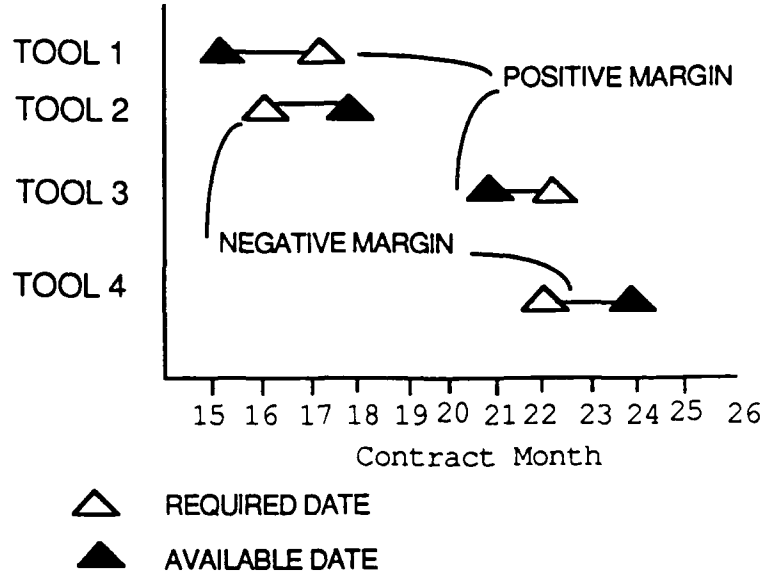


Figure (12): Software Development Tool Indicator, (Adapted from AFSCP 800-43)

Other Notes on SMI Usage

The previous section detailed six basic software management indicators and their supporting metrics. They represent a suite that is still evolving and adapting to the rapidly changing software development environment. It is imperative that they be customized or "tailored" to each project's dynamic management information needs.

Customizing of the basic suite may involve expanding or reducing metric measurements. A project manager may want to track utilization of both development and system computer resources, different types and skill levels of personnel, the length of time SPRs remain open, or other project unique attributes. What is essential is that the project

manager understand not only his project's information needs, but also what the information means.

The application and interpretation of SMIs is not clear cut and easy to "cookbook". The indicators must be compared and contrasted with each other to verify suspected trouble areas. For instance, if the Development Progress indicator shows trouble in the integration phase, the Software Manpower indicator may show an abnormal unplanned loss rate or shortage of experienced staff. Another situation might show the testing phase schedule delayed. The reason for this may be traced with the Software Development Tool indicator as a delay in the development of testing tools.

The point is that indicator metric measurements are all coming from the same development "system". Therefore, the indicators are interrelated in their causes and thereby their analysis and interpretation. Problems arise from the effects of the previously discussed risk drivers. Their impact will hopefully be manifested in SMI reports so that the information can be fed into the software risk management system shown in Figure 4 found in Chapter I.

The sophistication of the SMI reporting system will depend on the complexity of the project and the ability of the project staff to administer it. A simple project may require only a few indicators, where a complex project will require more extensive usage.

The nature of the metrics is such that they require a project manager to predict and report expected work difficulties selected from events which have already occurred. Given this, the "fresher" the data,

the sooner problems can be identified and resolved. The best systems will not keep project managers out of trouble, but will keep them from being surprised when trouble comes.

Conclusion

In this chapter the concepts and problems related to software management indicator usage were discussed. A fairly detailed description of a suite of indicators, gathered and developed by the U.S. Air Force, was presented as a "model" of an "early warning system" for software development risk management. The next two chapters will outline a study that was done to examine current SMI use in the defense industry.

CHAPTER III

EMPIRICAL STUDY

An exploratory empirical study was undertaken to qualitatively explore the implementation of Software Management Indicators in the defense industry. From what can been ascertained from preliminary research, little formal analysis and follow-up evaluation of SMI implementation has been done. One exception is a working paper by MITRE corporation for the USAF Electronic Systems Command (ESD-WP-27367, 1987). The report addressed the effectiveness of a set of indicators/metrics used on nine ESD projects. The results were used to improve the ESD indicators presented in the previous chapter.

Goals for this research presented in this study were to:

1. Seek evidence that supports or refutes the use of Software Management Indicators to help manage and improve the software development process.

2. Solicit comments relating to SMI policy and implementation issues, tangible and intangible benefits, problem areas, and recommendations for change.

3. Gain a better understanding and appreciation of the 'trials and tribulations' of managing large scale software development and acquisition and how the use of SMIs might apply.

4. Determine future research needs.

Data Collection Methodology

An interview process consisted of identifying and contacting government and defense contractor authorities who have managed large-scale defense software development projects and have used SMIs. Note the four contact types in Figure 13. They reflect the level of management/usage in which the contacts were found. Note also the types of organizations contacted. They represent the range of organizations found in the defense software industry.

Face-to-face and telephone interviews were conducted with these contacts and focused on the topics outlined in Appendix A. The topical discussions were purposely "open-ended" to allow an unstructured free exchange of information. This format was used because of the wide range of experience with the use of SMIs, and because of the exploratory/discovery nature of the study. As a result of this format, all of the topics outlined in Appendix A were not always covered completely and typically led to other pertinent and related information. The limitations of this data gathering methodology and approach will be addressed next.

Limitations of the Study

There were a number of limitations to gathering and compiling information on the implementation and use of SMIs. They include:

1. SMI use is immature, varied, and relatively new. With this newness, historical project data reflecting SMI use is lacking. Where

ORGANIZATION	CONTACT TYPE Systems/Software Engineer	Technician	Project Manager	Contract Admin/Audit
Defense Systems Management College, Software Engr Institute	3			
AF Special Project Offices (SPOs)	4			
Air Force Systems Integration Offices	2			
Defense Contract Admin Agencies (Defense Logistics Agency, AF Plant Representative Office)				6
Defense Contractors	5	3	3	1
Dept of Defense, AF Systems Command, and Product Divisions	5		4	4

Figure (13): Summary of the Empirical Study Interviews, Showing the Distribution of Contact and Organization Types and Numbers of People Interviewed.

SMIs were being used there was wide variation in the degree to which they were implemented. Use in a formal sense is very limited and immature and where present, is rather informal. This immaturity and variation made it difficult to correlate and verify use between similar organizations and types of projects.

2. Data gathering and scope was constrained by resources. In itself, the interview process was difficult to manage. The scope was limited to government and defense related organizations as previously indicated in Figure 13. Physically contacting these people either by

phone or in person consumed much more time and resources than initially estimated. The result was a limit to the number of and the degree to which interviews were conducted. Just identifying those who have used or are specifically familiar with the formal Air Force SMI concept, proved more difficult and limited than originally anticipated. Some of the difficulty encountered may reflect a firm's desire not to discuss management systems that may be giving them a competitive edge.

3. Data gathering was inconsistent and discontinuous. When interviewees were finally contacted, interviews were conducted at their whim. It was particularly difficult to get many individuals to spend any length of time discussing the subject matter. The most knowledgeable contacts were typically very busy and had little time to "chat" about the successes they were having. Because of this and of the partial familiarity with SMIs, many times it was very difficult maintaining continuity in the interview data gathering process. Therefore much of the information presented is fragments of conversations and comments that have been pieced together.

Given this, it is hard to trace many observations to unique sources. Though not gathered in a very rigorous manner, the observations have notable value in qualitatively yielding a "conventional wisdom" view of SMI use and success. This "conventional wisdom" view will be presented in the next chapter. Future research may try to quantitatively verify these observations.

4. The observations are heavily biased toward those organizations who use SMIs. This study purposely targeted organizations that were identified as users of SMIs. Identifying organizations using SMIs was difficult enough, trying to find organizations who may have tried a formal SMI approach and has since found them unuseful proved impossible. The later case is once again due to the immaturity of SMI use; many firms have barely begun using them and therefore have not had time to evaluate the results.

5. Opinions of decision makers were difficult to obtain. SMIs are intended for top management's use and basically present a project management overview from which they make critical decisions. Even so, it was difficult to find top managers familiar with SMIs at any depth. The typical level of significant knowledge and insight was found in a non-management, "quality control" type function that was fairly technical and production oriented. Viewpoints at this level typically reflect an interviewee who generates the SMI data, but does not make decisions from it. SMIs were devised to help top managers make project management control decisions. But , as observed, it was almost as if management wanted SMIs used because they were "supposed to" (because of government suggestions), without understanding why.

The following chapter will synthesize the results from this exploratory data collection process.

CHAPTER IV

RESEARCH FINDINGS

This chapter will present a summary of exploratory research findings. An intuitive, "Conventional Wisdom", summary will be offered, as will more "hints" and suggestions for SMI implementation and usage. This Conventional Wisdom approach merely represents the observations and information gathered in research as they generally seem to apply to the various issues associated with SMI concepts and usage.

SMI Conventional Wisdom

1. When used, SMIs provide greater visibility and control of the software development process. Comments gathered in interviews with a variet y of government authorities suggest many defense contractors don't have an effective system to systematically manage software development. Surprisingly, even modest use of software engineering and structured development techniques by some government contractors was notably lacking.

Tangible cost and benefit data is difficult to collect. Generally speaking, even though collection and processing indicator data is costly, most people interviewed seem to feel the benefits received were worth the cost. The most common intangible benefits cited by project managers was, "at least now I know what my people are doing" and

"they may not be perfect, but things are certainly better than what we had before". By being required or "coerced" into using SMIs, many contractors have come to realize their value.

Putting dollar amounts on the value of these intuitive feelings, like most information systems, may prove impractical. If the increase in the use and development of costly automated indicator control systems is any indication, then there must be some cost effective benefit being derived.

From another viewpoint, when the indicators are used to "avoid" expensive cost risk drivers, it only takes "dodging one bullet" before the cost is quickly justified. This is the "pay me now or pay me later" philosophy at work.

2. The concept of software risk management is not well developed. Throughout this study, risk management and quality control were used as a conceptual and practical framework for SMI use. In discussing the use of SMIs in relation to this framework, not one contact indicated a "formal" systematic use of the indicators in a risk management process. Though recognized as a method to manage software project management risk, no well developed planning and control risk management system that methodically included them as input was found. There was evidence in a couple of organizations that this level of "maturity" was being approached, but still had a way to go.

3. SMI use is in an "early adopter" stage of implementation. The relative immaturity of incorporating the indicators into a formal risk management process seems to reflect the relative immaturity of the

indicators. Only recently (1980s) have defense contractors begun to formally and systematically implement software "management" indicators. Certainly there have been "pioneers" in their use (e.g. TRW Systems), but until recent years have they only "diffused" to the "early adopters".

Exposure to the indicators seems to be hampering their diffusion. The number of people who know about, or better yet, know how to use SMIs is relatively small. Most organizations interviewed (government and industry) had only a very select and minute staff (many times one or two deep) that had any practical knowledge of the gathering, use, and interpretation of indicator data.

In most cases discussed, training and use of the indicators is bottom-up vs. top-down. As devised, the fundamental use of the indicators was for "top" management to get a broad overview of their software projects. This is not always the case. It may be that the managers don't know what they need in the way of software development control feedback. "They don't know what they don't know". A common comment was that systems level project managers don't know "anything" about software, and aren't about to micromanage it. They will continue to expect the software will just "show up" when it is needed to integrate into the overall system.

Given the trouble in making contacts, there may be a question of whether these companies have better, "proprietary", methods they use and aren't willing to discuss. If so, research findings did not conclusively verify this. It can be said that TRW Systems, a pioneer in software

metrics, is actively using SMIs that closely conform to the Air Force model.

4. Managers who are now using SMIs are doing so enthusiastically. Most of those who are now adopting the indicator approach seem to be "trying them on for size" after hearing of their benefits and use from the "pioneers". Generally, contractors and government project managers are a bit skeptical about having to collect more costly data that may be worthless or even "sensitive" in the hands of industry competitors. Those who are now using them regularly are doing so carefully, but enthusiastically.

This study did not attempt to estimate the percentage of defense contractors using "formal" SMIs. Intuitively, it's felt the percentage of contractors using SMIs would be less than 30 percent.

5. SMIs must be implemented in a proper environment. The use of the SMI approach "flies in the face" of the way many government contractors have previously done business. The "trust me" syndrome present in government procurement is well ingrained (and probably more so in the commercial sector). SMIs, when assumed effective, document contractor performance. With increased public and Congressional pressure to account for contractor performance, there is a tendency for contractors to avoid and resent producing potential "evidence" of poor performance. Also, with more and more government contracts taking a harder look at past performance, contractors will have greater concern for how "follow-on"use of SMI data will impact them.

Projects are far more vulnerable to dishonesty than the ongoing operations. Standard operations are characterized by considerable knowledge about expected system performance. When the monitoring system reports information that deviates from expectations, it is visible, noteworthy, and tends to get attention.

In the case of many projects, expectations are not so well known. Deviations are not recognized for what they are. The project manager is often dependent on team members to call attention to problems. To get this cooperation, the "bearer of bad news" is not punished; nor is the "admitter-to-error" executed. On the other hand, the "hider-of-mistakes" may be shot with impunity (Meredith and Mantel, 1985, p.291).

There is some tendency for project monitoring systems to include an analysis directed at the assignment of blame. While the managerial dictum "rewards and punishments should be closely associated with performance" has the ring of good common sense, it is actually not good advice (Meredith and Mantel, 1985, p. 292). In other words, rewards and/or punishment based on a measure of performance may prove to have a negative effect on potential productivity.

All of this is to say, an environment that allows a myopic, short-term viewpoint is not apt to motivate people to better performance. The practice is apt to result in lower expectations. If achievement of goals is directly measured and directly rewarded, a tremendous pressure will be put on people to understate goals and to generate plans that can be met or exceeded with minimal risk and effort.

In this study, the programs that were held as the 'finer" examples, were the ones in which the climate allowed for short term problems to be resolved without punishment for those responsible. A "long term" view was taken in which "bad" feedback on project progress was used to stimulate corrective action and not to "crucify" the project manager for not doing his job. This environment is typical of that found in many of today's more successful companies.

Also present in these environments was an independent group who gathered and reported indicator data. The effect of this approach is that the group serves both the project managers and the developers. This approach is an attempt to promote integrity, prevent bias, and expedite reporting of "bad" news. In operation, the group would brief the developer project manager before briefing top management. This not only keeps the development staff better informed, but also gives them time to respond to problems that are being identified.

This section presented various observations and findings from the exploratory research study. The next section will offer some hints on implementing SMIs found in this research.

Implementation Hints

1. The use of SMIs can give a project manager a false sense of security. By strictly looking at SMI results a project manger may "assume" that all is well with the development process. This may be, but this assumes the requirements were right in the first place.

If there is one over-riding implementation issue and problem, it has to be lack of a stable, well-defined requirements. This is nothing

new. Requirements analysis is the most crucial step in the development of a software system. There must be a sufficient requirements baseline. If not, a perfectly managed development process will only produce a perfectly built system that doesn't meet the user's true needs.

2. SMIs provide the early feedback necessary to re-assess planning and yield "convergence" between estimates and reality. A project management baseline is only as good as the tools used to develop it. Despite estimating tools such as COCOMO (Boehm, 1981), better tools and approaches for developing project estimates need development. This problem is well recognized, with improvements actively being pursued. One thing that must be remembered, no matter how good the estimate, there is still some percentage of uncertainty that results from software risk drivers. Theoretically, as a project gets closer a planned objective, the better the understanding of the project's risk and estimates to completion. It must be assumed that at project completion the estimate will be perfectly correct. Project management must continually re-assess and manage this project risk.

3. As implemented, the SMIs aren't overwhelming with the amount of data produced, and this should remain a goal. One of the more distressing trends in project management is the tendency to overstress control (Meredith and Mantel, 1985). Various types of project management information systems have been developed recently which can provide an abundance of data concerning a project. Many of the information systems extend through many management levels and require the project participants to report a large amount of data.

When questioned whether SMIs may be overstressing control, most people felt it wasn't. Most felt it was control that has been needed, and that more appropriate control may help the software management problem.

4. SMIs should be and need to be "tailored" to each project's unique needs. Another trend is the current tendency to rely heavily on complicated, sophisticated project management systems. There is a very real danger that project managers can become so preoccupied with the system that they fail to exercise enough personal management of a project (Cleland and King, 1983). Risk management and control is of a personal nature, so it is important that project managers use techniques that reflect their personality and are consistent with the complexity of the project.

5. The mechanism that allows the sharing of changes in and development of new techniques should be developed and maintained. Organizations already using some form of indicators tend to promote and offer some they use "internally". This is fine and has been the source of many indicators that have been "adopted" by the Air Force. There is a need to accommodate and incorporate virtues of the variety of indicator "systems" that are evolving. Maybe a "user group" of sorts could be created.

6. Implementation of SMIs should reflect a conservative viewpoint. Gilbreath (1986) warns of the improper use of project data. He states that no matter how many additional channels of communication that are made available and accessible, the time

available to examine the data remains the same. His perception is that as more data is available to more and more people the more susceptible it is to misunderstanding, misuse and confusion. Like any use of data, he argues, data has a very special role, but its role should be limited and kept in perspective. Sometimes more or better data has little or no value as information.

Gilbreath's professions basically warn to "Keep It Simple Stupid". Projects are complex entities that have to be managed with complexity. For the "normal" project manager to comprehend the status of his project, he needs the minimum of information versus maximum. .

7. Approach implementation like a "real" project. After nearly a decade of professing the virtues of structured development techniques, Ed Yourdon (1979) was moved to write "The Second Structured Revolution". His dissertation is an attempt to create "born again" developers who would actually use the structured techniques. In his opinion the failure of the "first" structured revolution was due to management. Observations and lessons learned from his analysis may prove worthwhile when looking at the implementation of Software Management Indicators.

In looking at why the "first" revolution hadn't proved successful, he cited: inadequate selling of the techniques; inadequate training; and inadequate management and follow-through. The solution he advocates is to approach implementation the same way a "real" project should be approached. Essentially he is suggesting that improving software projects is a project in itself.

What's needed, he suggests, is a "dignified" selling effort, and formal analysis, design, implementation and testing activities. Each level within the organization must be approached differently. Top management needs economic justification. Middle management will want to know how much time and how many people it will take., The technicians will be suspicious of another management program. The best approach may be a top-down approach.

Analysis activities are used for what basically amounts to a feasibility study. Questions such as: do we have a management problem?; what is the nature of the problem?; is it politically sensitive?; and what are the risks of using the indicators?; should be asked. The Design phase is the planning phase. Here questions might be: which indicators should be used?; which project should they start on?; how do we manage the implementation process?; and how do we measure the impact?.

Here it is suggested to limit the number of techniques introduced at once; to select a moderate sized project; to groom some "gurus"; and to establish a group that coordinates introduction and training of the new techniques. Implementation should be considered an evolutionary process. New indicators should be introduced and old ones eliminated as needed. Testing activities are used to continually monitor the worth of the indicators and to recommend changes.

The most important point made by Yourdon is that the structured techniques are not "magic". This is most true for Software Management Indicators. As with structured techniques, the indicators

represent "standardized common sense". To effectively implement SMIs, there must be a systematic approach.

Conclusion

This chapter presented pertinent findings from the exploratory research conducted for this project. Several hints on SMI implementation and usage were offered. The next chapter will briefly summarize these findings and offer some future research directions.

CHAPTER V

SUMMARY AND CONCLUSION

This study suggested the need for an "early warning" system to help manage the risk inherent in software development project management. The fundamentals of metric measurement and indicator use were presented as foundation for the use of software management indicators as such a system A suite of Software Management Indicators currently implemented by the U.S. Air Force was described to serve as a model of the "state of the art" in software indicator management.

In this chapter pertinent findings will be summarized and then some recommendations for future studies will be discussed.

Summary of the Research Findings

As software project managers consider implementation of SMIs, they may want to reflect on some thoughts from Robert Pirsig's (1974) Zen and the Art of Motorcycle Maintenance:

> To speak of certain government and establishment institutions as the system is to speak correctly, since these organizations are founded upon the same structure conceptual relationships as a motorcycle. They are sustained by structural relationships even when they have lost all other meaning and purpose. People arrive at a factory and perform a totally meaningless task from eight to five without question because the structure demands that it be that way. There's no villain, no "mean" guy who wants them to live meaningless lives, it's just that the structure demands it and no one is willing to take the formidable task of changing the structure just because it is meaningless.

> But to tear down a factory or to revolt against a government or to avoid repair of a motorcycle because it is a system is to attack effects rather than causes: and as long as the attack is upon effects, no change is possible. The true system, the real system, is our present construction of systematic thought itself, rationality itself, and if a factory is torn down but the rationality which produced that government are left intact, then those patterns will repeat themselves in the succeeding government. There's so much talk about the system. And so little understanding.

There are many reasons why the management of the software development process is difficult. It is imperative that systems developers continue to examine the "structure" underlying the software development "system". Better "understanding" of this structure is necessary so that new systematic and rational approaches to management can be developed. The use of Software Management Indicators is one such approach.

Requirements definition is still the biggest problem facing software development managers. The system must be understood before it can be modeled and changed into a software system. SMIs do provide greater visibility and control into this software development process, but are not the total means to the end, just a part of it. Primarily they are used merely to stimulate further management action within the context of software risk management.

SMIs are in an "early adopter" stage of acceptance. Initial use of SMIs has been driven by contractual requirements and/or effective selling by the government. Contractors are enthusiastically embracing the SMI concept. After getting past the initial phases of implementation, they seem to working to improve the fundamentals put forth by the Air Force. Use is not just for the benefit of the government. The contractors

interviewed have found they enhance their internal management process.

Incorporation of SMIs into a formal software risk management process was not found. Findings indicate the indicators are only used in an informal risk management sense.

Format is not important. Customizing of SMIs to unique project needs is key. Not all indicators are always appropriate. Better automation of the SMI process is a must. Though beneficial, collection, analysis, and reporting SMI data is costly and time consuming.

Continued assessment and enhancement of current indicators and further incorporation of new ones must be done systematically. A process of gathering feedback, and "sharing" information may be needed.

SMIs are specifically looking at the quality of the process of developing software, they must be used in conjunction with other techniques that look at the quality of the software product. In essence this is a "Total Quality Control" approach.

Recommendations for Future Research

Future SMI work may broaden to include exploration of usage in the commercial software development environment. Further, broader, more detailed and quantitative work is certainly warranted in the defense business given there is greater resources and better access to sources of historical data from SMI use. Access to such sources will be administratively and logistically difficult and costly for both environments.

It is recommended that there be authority and sponsorship from a noteworthy organization that gives greater incentive to those cooperating in time-consuming research.

More formal incorporation of the SMIs into the software risk management process should be studied. The primary formal use of SMIs is in the process, but is currently very immature. Better understanding of the risk management and "total quality control" approaches in software development are sorely needed.

This work has merely "scratched the surface" of work needed in software project management. The challenge is for software development management to acknowledge and to continue the evolutionary process needed not only to improve the quality of the development process, but also its management. It is hoped future researchers and developers will recognize and accept this challenge, so to better manage the risks of our profession.

REFERENCES

AFSCP 800-45, Air Force Systems Command and Air Force Logistics Command, Software Risk Abatement, Andrews AFB, DC, unpublished draft.

AFSCP 800-43, Air Force Systems Command, Software Management Indicators, Department of the Air Force, Andrews AFB, DC, January 1986 and unpublished draft revision.

Birrell, N.D. and M.A. Ould, A Practical Handbook for Software Development, Cambridge Univ. Press, Cambridge, 1985.

Boehm, Barry W., et al., "Quantitative Evaluation of Software Quality", Proc. 2nd International Conference on Software Engineering, October 1976, pp. 592-605.

Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

Boehm, Barry W., et al., Characteristics of Software Quality, North Holland Publishing Co., New York, New York, 1978.

Cleland, David I., and William R. King, Systems Analysis and Project Management, McGraw-Hill, New York, NY, 1983.

Cooper, J.D., "Corporate Level Software Management," IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, July 1978, quoted by Tarek K. Abdel-Hamid and Stuart E. Madnick, "Lessons Learned from Modeling the Dynamics of Software Development", unpublished paper, Aug 1988.

Davis, Bob, "Costly Bugs: As Complexity Rises, Tiny Flaws in Software Pose a Growing Threat", Wall Street Journal, January 29, 1987.

DSBTF: Defense Science Board Task Force, Report of the DSBTF on Military Software, Office of the Under Secretary of Defense for Acquisition, Washington, D.C., September 1987.

Demarco, Tom, Controlling Software Projects--Management, Measurement, and Estimation, Yourdon Press, New York, NY, 1982.

Denicoff, Marvin, and Robert Grafton, "Software Metrics: A Research Initiative", in Perlis, A., et al., (eds.), Software Metrics: An Analysis and Evaluation, MIT Press, Cambridge, Massachusetts, 1981, pp.13-18.

DOE: Department of Energy, DOE/CR-0015, Cost and Schedule Control Systems Criteria for Contract Performance: Implementation Guide, May 1980.

ESD-TR-88-001, Air Force Systems Command; Electronic Systems Division, Software Management Indicators, Department of the Air Force, Hanscom AFB, Massachusetts, May 1988.

ESD-WP-27367, Air Force Systems Command; Electronic Systems Division, An Initial Evaluation of Metrics Reporting on ESD Programs, Department of the Air Force, Hanscom AFB, Massachusetts, May 1987, p. 9.

Gilb, Tom, Software Metrics, Winthrop Publishers, Cambridge, Massachusetts, 1977.

Gilbreath, Robert D., Winning at Project Management, Wiley, New York, New York, 1986.

Jones, Capers, Programming Productivity, McGraw-Hill, New York, New York, 1986, pp. 5-6.

Juran, Joesph M. and Frank M. Gryna, Jr., Quality Planning and Analysis, McGraw-Hill, New York, New York, 1980.

Kosarago, S., and H. Ledgard, "Concepts in Quality Design", National Bureau of Standards Technical Note 842, August 1974.

Meredith, Jack R., and Samuel J. Mantel, Project Management: A Managerial Approach, Wiley, New York, New York, 1985.

Orr, Ken, "Managing the Software Crisis," Computerworld, July 15-22, 1985.

Page-Jones, Melir, Practical Project Management: Restoring Quality to DP Projects and Systems, Dorset House, 1985.

Pirsig, Robert, Zen and the Art of Motorcycle Maintenance, Bantam, New York, New York, 1974.

Rubey, R., and R. Hartwick, "Quantitative Measurement of Program Quality", Proc.23rd National Conference, ACM, 1968, pp. 671-677.

TRW-SS-73-09, Boehm, et al., Characteristics of Software Quality, TRW Systems, One Space Park, Redondo Beach, California, 1973.

Weinberg, G.M., and E.L. Schulman, "Goals and Performance in Computer Programming," Human Factors, 1974, 16(1), pp. 70-77.

Yourdon, Edward, "The Second Structured Revolution", Yourdon Inc., New York, New York, 1979.

Zmud, R. W., "Management of Large Software Development Efforts," MIS Quarterly, Vol. 4, No.2, June 1980, pp. 45-56.

APPENDIX A

INTERVIEW TOPICS

Captain Court C. Allen
USAF/University of Colorado
Master of Science; Management
Science and Information Systems

Thesis: Software Management Indicators: Managing the Risk of Software Project Management

INTERVIEW GOALS:

1. Seek evidence that supports or refutes the use of Software Management Indicators (SMIs) to help manage and improve the software development process.

2. Solicit comments relating to Software Mangement Indicator policy and implementation issues, tangible and intangible benefits, problem areas, and recommendations for change.

3. Gain a better understanding and appreciation of the 'trials and tribulations' of large scale software development and acquisition.

INTERVIEW TOPICS:

1. A Software Crisis has been declared. With the advent of software engineering principles and techniques, the technical methods of software development have improved notably. Unfortunately, management methods have not.

o Do you share this view ? If so, what are some of the management problems as they tend to apply to the DOD software acquisition process.

2. Many 'large-scale' DOD acquisition programs use 'traditional' project management techniques (WBS, PERT/CPM, Earned Value, etc.).

o Is it your experience that these techniques are being used to manage the software development process?

o As they apply to software development, are these techniques appropriate? Is there a better way?

3. By definition an indicator is a sign or symptom of something else. Software Management Indicators have been offered as a means to gain greater visibility and control into the software development process.

o How would you define visibility and control?

o In your experience do these indicators offer this?

o Are they just another example of overstressing control at the expense of productivity or is productivity ultimately enhanced?

o Are we collecting meaningless data? Are we overcollecting data?

4. In Tom DeMarco's book, Controlling Software Projects, he offers 'The Metrics Premise': Rational, competent men and women can work effectively to maximize any single observed indication of success. He says if you indicate some clear quantitative definition of success, (e.g. program size, data space used, completion time, etc.) and if you measure and track progress toward that success, project members will align themselves with the stated goals.

o Do you find this to be true? Do contractors tend to use two 'books' when reporting project status; one internal and one for the government?

o Do production/development staff workers resent tracking their progress?

5. Risk management has been deemed essential to dealing with uncertainty inherent in management of large projects. It seems important that a risk management strategy program be established early and continually managed throughout a project's lifecycle. Risk monitoring requires that risk be recognized and acted upon if discovered.

o Is formal software risk management actively used in your software development projects?

o Do you feel SMIs are an effective tool in the risk management process?

o Typically, are results from indicators directly tied to contingency plans developed through the risk management process? If not, how do you manage projects where data indicate they may be going bad.

6. SMIs offer an 'executive' overview of project status. As a rule, do workers at the production level (systems analysts, designers, programmers, etc) have access to the indicator data concurrently for prospection or is it typically retrospectively late? In other words, is the data timely enough to allow changes to be made before it's too late?

7. Collecting and reporting data on the progress of a particular contractor's project can go overboard.

o Is it felt the government is asking for too much data?

o Government contracting is highly competitive. How sensitive is this data in terms of what a competitor can find out about a fellow contractor? In terms of what the government finds out about a contractor's internal management?

o Is data usually extracted from an existing 'internal' project management information system or is it collected ad hoc for the government?

8. Data collection, analysis, and reporting can be very costly.

o What are some of the tangible and intangible benefits of incorporating such effort?

o Can these benefits be measured?

9. What are some of the policy and implementation issues that you perceive with the use of these indicators? Other problems?

10. In general, do have any more comments or recommendations about the use of Software Management Indicators? About the management of software development projects?